EXTENDING THE REACH OF PARALLEL COMPUTING WITH CUDA

Mark Harris, NVIDIA

@harrism

#NVSC14

EXTENDING THE REACH OF CUDA

1 Machine Learning

2 Higher Performance

3 New Platforms

4 New Languages

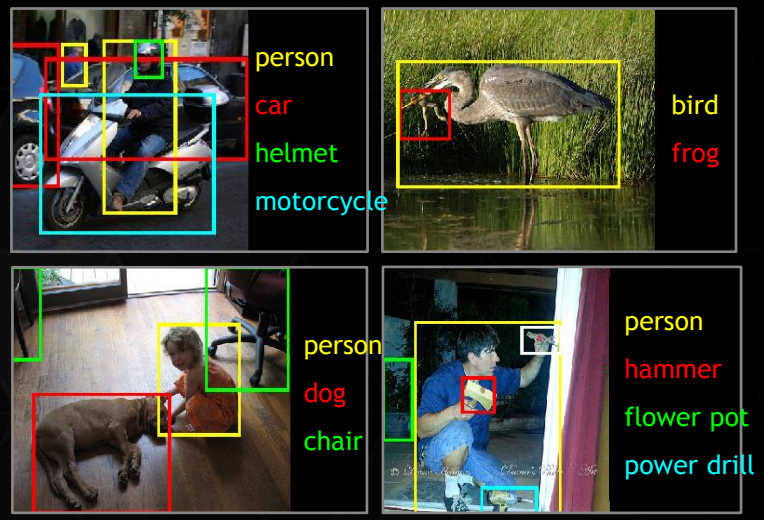
GPUS: THE HOT MACHINE LEARNING PLATFORM

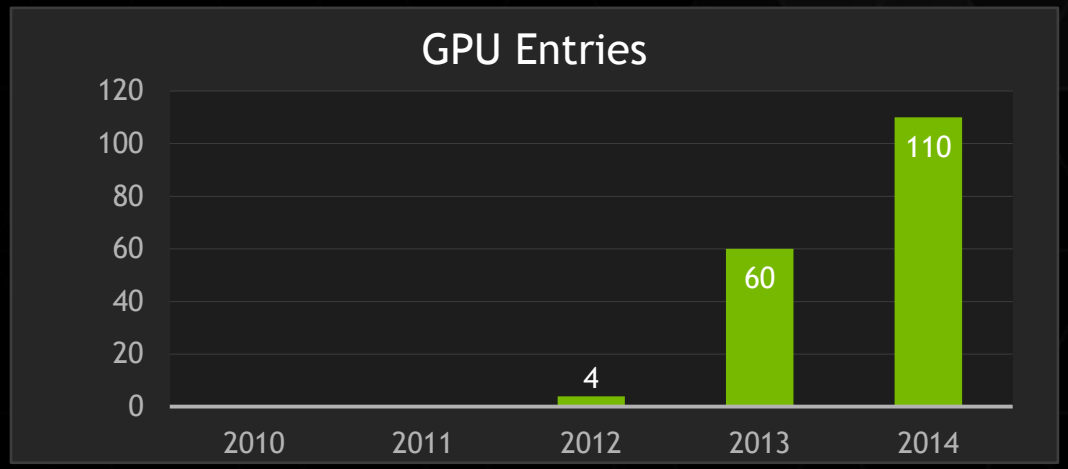
Image Recognition Challenge

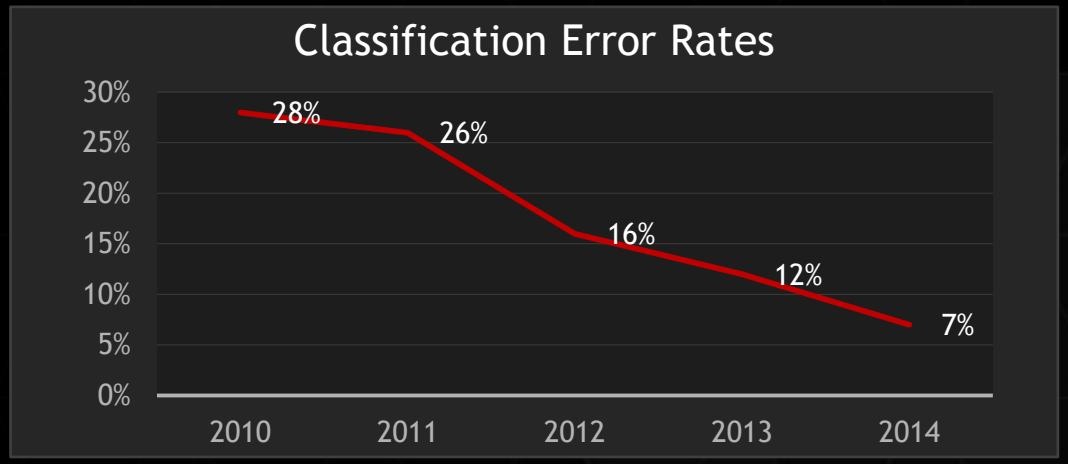
1.2M training images • 1000 object categories

Hosted by

IMAGENET



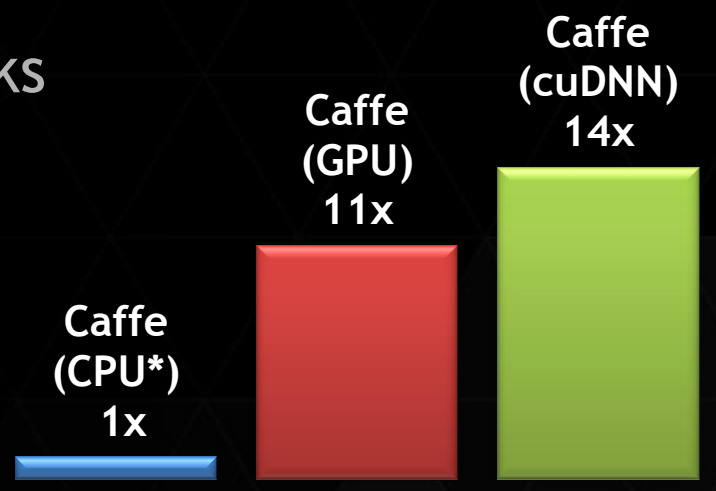




cuDNN GPU-ACCELERATED DEEP LEARNING

High performance routines for Convolutional Neural Networks

- Optimized for current and future NVIDIA GPUs
- Integrated in major open-source frameworks
  - Caffe, Torch7, Theano
- Flexible and easy-to-use API
- Also available for ARM / Jetson TK1
- https://developer.nvidia.com/cuDNN



Baseline Caffe compared to Caffe accelerated by cuDNN on K40

*CPU is 24 core E5-2697v2 @ 2.4GHz Intel MKL 11.1.3

EXTENDING THE REACH OF CUDA

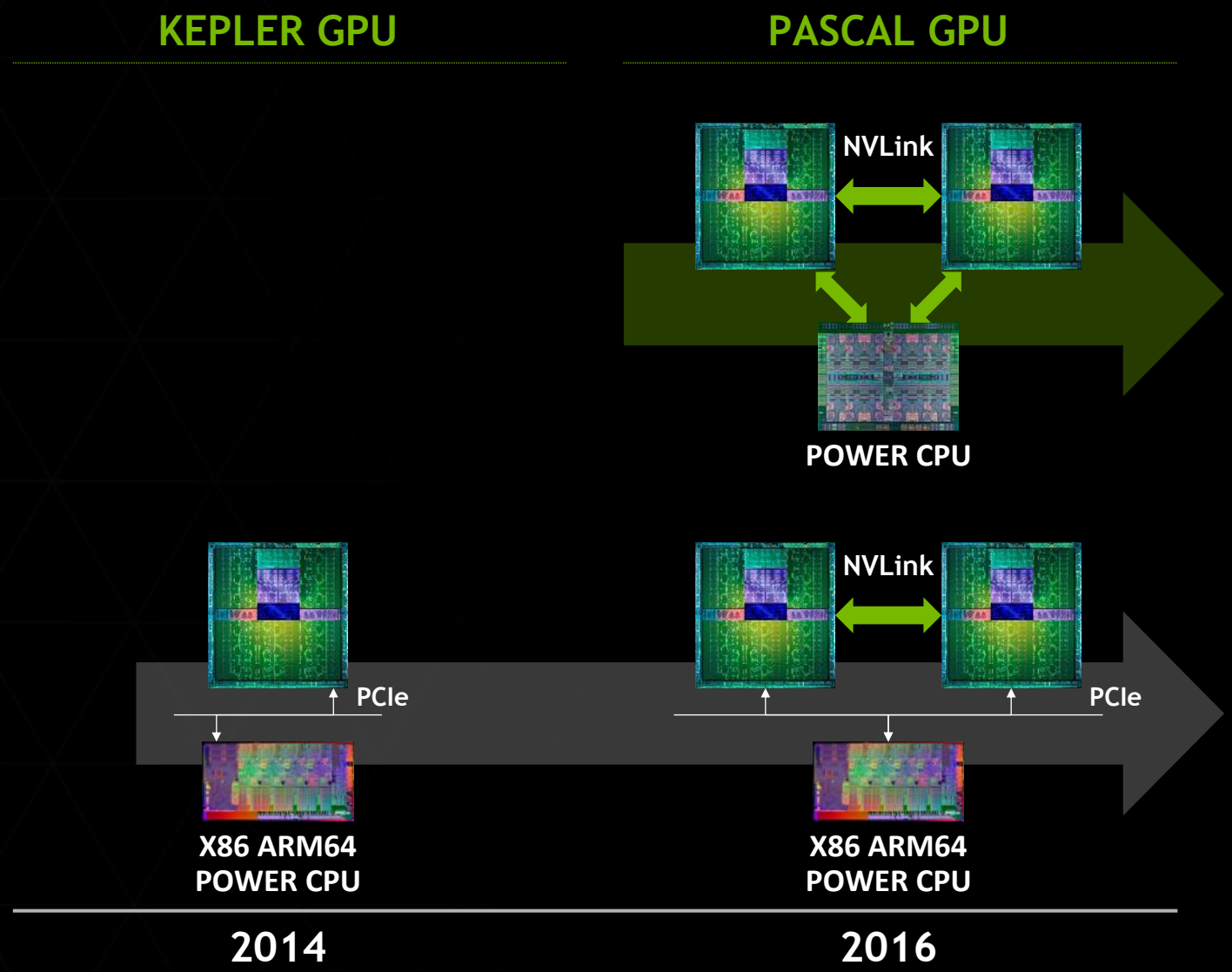
1 Machine Learning

2 Higher Performance

3 New Platforms

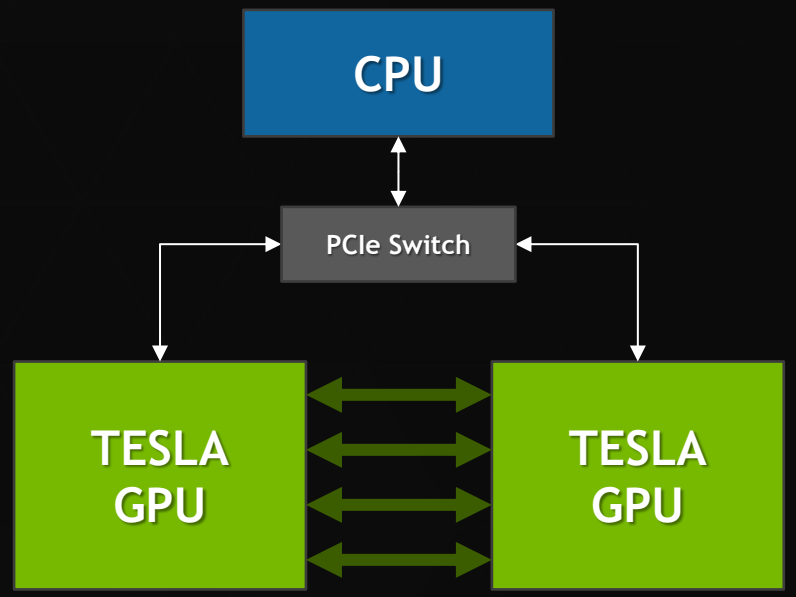
4 New Languages

NVLINK
HIGH-SPEED GPU INTERCONNECT



NVLINK UNLEASHES MULTI-GPU PERFORMANCE

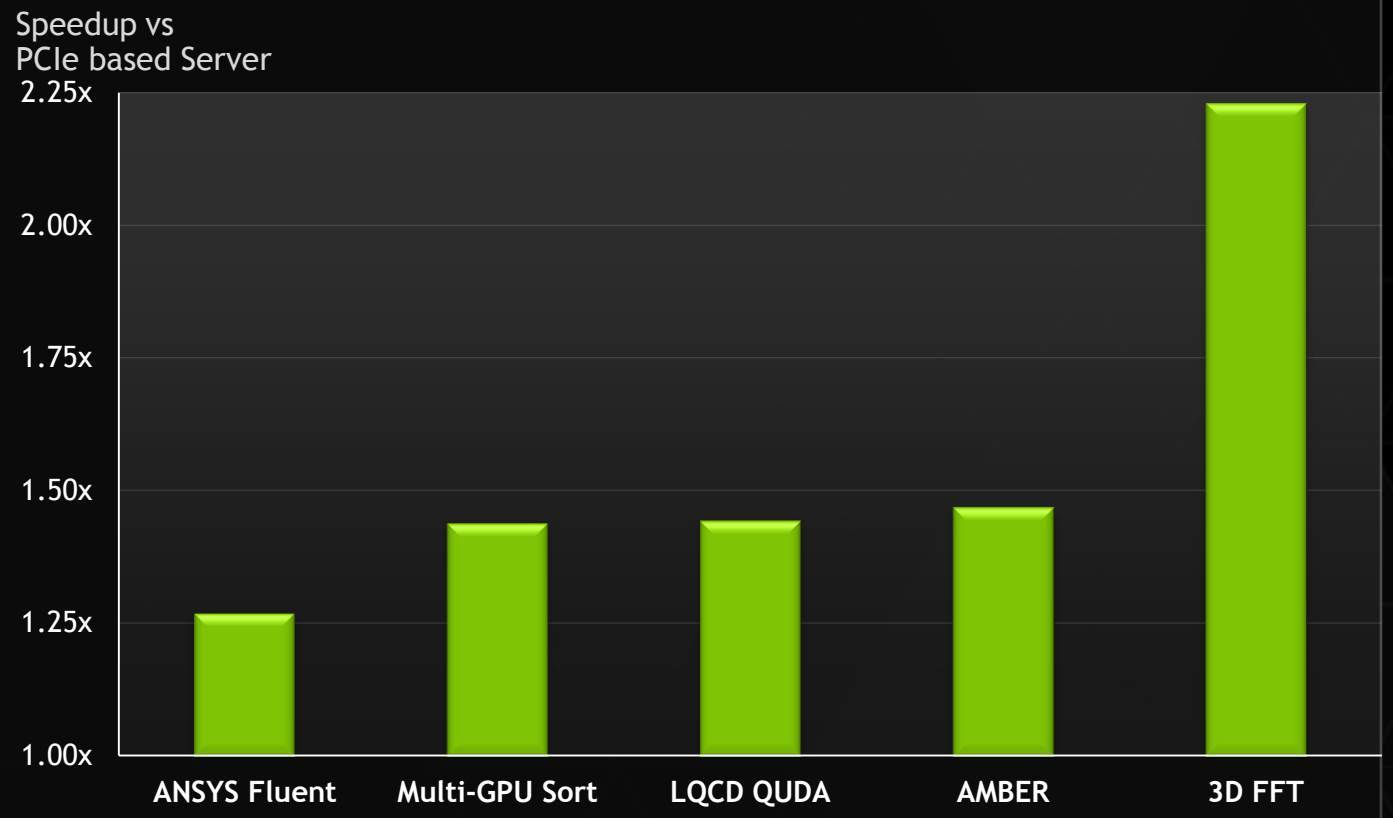
GPUs Interconnected with NVLink



5x Faster than PCIe Gen3 x16

Over 2x Application Performance Speedup

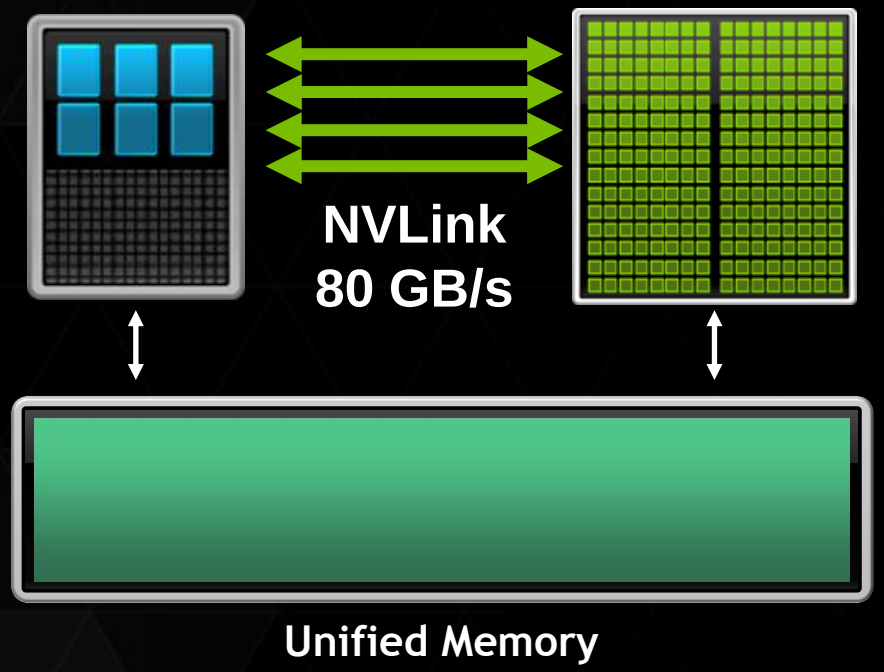
When Next-Gen GPUs Connect via NVLink Versus PCIe



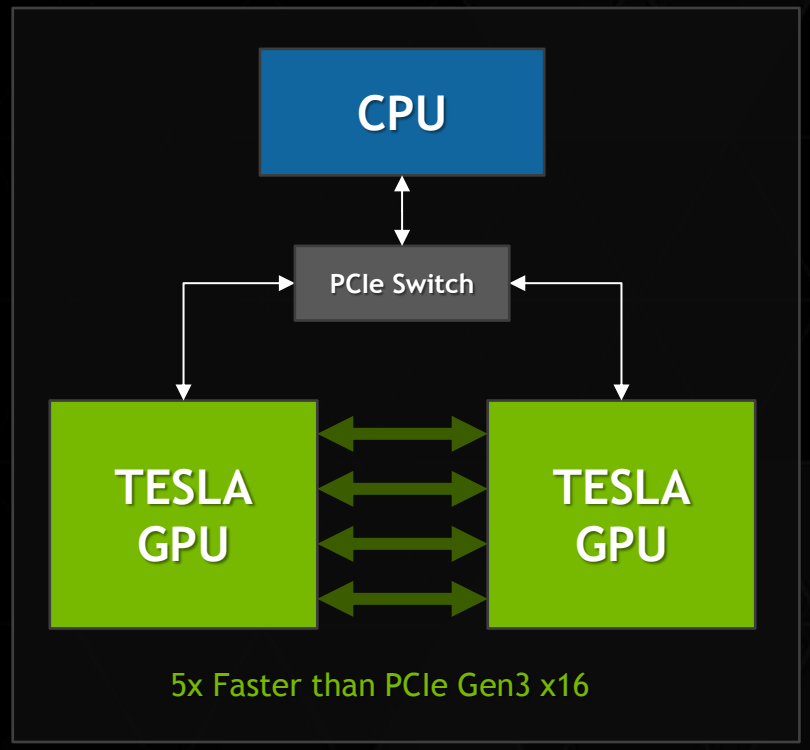
3D FFT, ANSYS: 2 GPU configuration, All other apps comparing 4 GPU configuration
AMBER Cellulose (256x128x128), FFT problem size (256^3)

NVLINK + UNIFIED MEMORY

Simpler, Faster



Share Data Structures at CPU Memory Speeds, not PCIe speeds



Eliminate Multi-GPU Scaling Bottlenecks

EXTENDING THE REACH OF CUDA

1 Machine Learning

2 Higher Performance

3 New Platforms

4 New Languages

Data Center Infrastructure

System Solutions
CRAY
DELL
IBM
hp
amazon webservices EC2

Communication
Mellanox TECHNOLOGIES
MVAPICH
OPEN MPI

Infrastructure Management
Adaptive COMPUTING
Bright Computing
IBM PLATFORM COMPUTING

Development

Programming Languages
C/C++
Fortran
OpenACC
python

Development Tools
allinea DDT
PGI
VAMPIR

Software Solutions
Kitware
MATLAB
ROGUE WAVE SOFTWARE

GPU Accelerators
GPU Boost
…

Interconnect
GPU Direct
NVLink
…

System Management
NVML
…

Compiler Solutions
LLVM
…

Profile and Debug
CUDA Debugging API
…

Libraries
cuBLAS
…

Tesla Accelerated Computing Platform

COMMON PROGRAMMING APPROACHES

Across a Variety of Heterogeneous Systems

Libraries

AmgX, cuDNN, cuBLAS, OpenCV, Thrust

Compiler Directives

OpenACC

Programming Languages

C/C++, Fortran, python, Java

EXTENDING THE REACH OF CUDA

1 Machine Learning

2 Higher Performance

3 New Platforms

4 New Languages

MAINSTREAM PARALLEL PROGRAMMING

- Enable more programmers to write parallel software
- Give programmers the choice of language to use
- Embrace and evolve key programming standards

MAINSTREAM PARALLEL PROGRAMMING

- Enable more programmers to write parallel software
- Give programmers the choice of language to use
- Embrace and evolve key programming standards

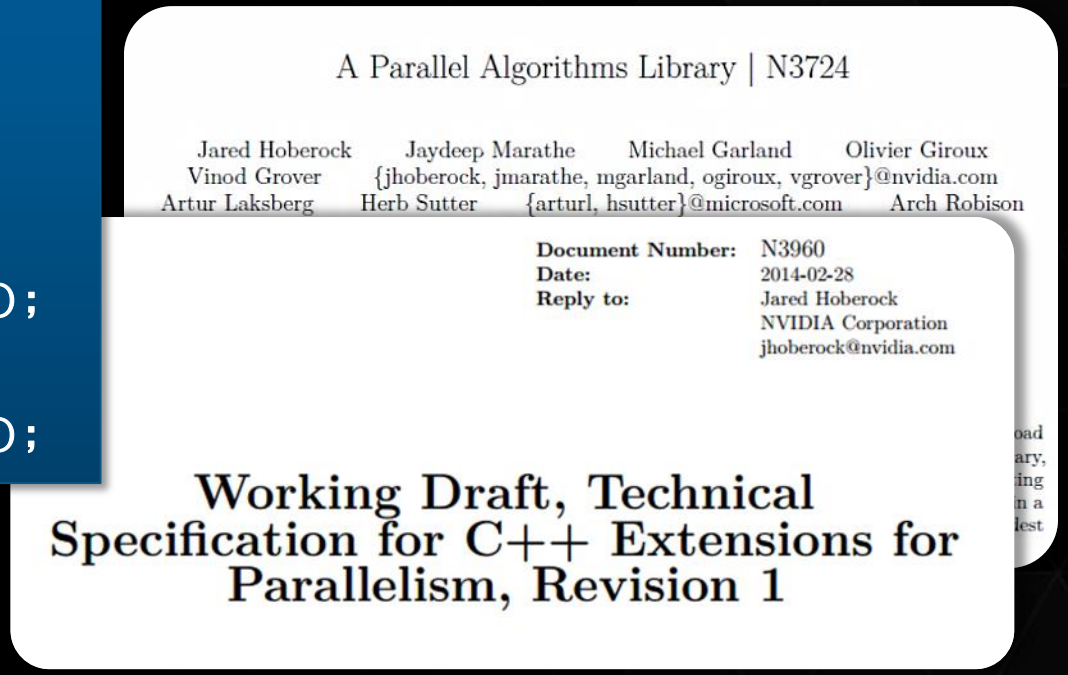
C++ PARALLEL ALGORITHMS LIBRARY PROGRESS

```
std::vector<int> vec = ...

// previous standard sequential loop
std::for_each(vec.begin(), vec.end(), f);

// explicitly sequential loop
std::for_each(std::seq, vec.begin(), vec.end(), f);

// permitting parallel execution
std::for_each(std::par, vec.begin(), vec.end(), f);
```


A Parallel Algorithms Library | N3724

Jared Hoberock Jaydeep Marathe Michael Garland Olivier Giroux
Vinod Grover {jhoberock, jmarathe, mgarland, ogiroux, vgrover}@nvidia.com
Artur Laksberg Herb Sutter {arturl, hsutter}@microsoft.com Arch Robison

Document Number: N3960
Date: 2014-02-28
Reply to: Jared Hoberock
NVIDIA Corporation
jhoberock@nvidia.com

Working Draft, Technical Specification for C++ Extensions for Parallelism, Revision 1

- Complete set of parallel primitives: for_each, sort, reduce, scan, etc.
- ISO C++ committee voted unanimously to accept as official tech. specification working draft

N3960 Technical Specification Working Draft:
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n3960.pdf
Prototype:
https://github.com/n3554/n3554

Linux GCC Compiler to Support GPU Accelerators

Open Source
GCC Effort by Mentor Embedded

Pervasive Impact
Free to all Linux users

Mainstream
Most Widely Used HPC Compiler

mentor embedded

On Track for GCC 5

> *"Incorporating OpenACC into GCC is an excellent example of open source and open standards working together to make accelerated computing broadly accessible to all Linux developers."*
>
> Oscar Hernandez
> Oak Ridge National Laboratories

OAK RIDGE National Laboratory

OpenACC Directives for Accelerators

NUMBA PYTHON COMPILER

Free and open source JIT compiler for array-oriented Python

- numba.cuda module integrates CUDA directly into Python

```
@cuda.jit("void(float32[:], float32, float32[:], float32[:])")
def saxpy(out, a, x, y):
    i = cuda.grid(1)
    if i < out.size:
        out[i] = a * x[i] + y[i]

# Launch saxpy kernel
saxpy[100, 512](out, a, x, y)
```

- NumbaPro: commercial extension of Numba
  - Python interfaces to CUDA libraries
- http://numba.pydata.org/

python

CONTINUUM ANALYTICS

ACCELERATING JAVA 3 WAYS

IBM

Drop-In Acceleration

```
java.util.Arrays.sort(int[] a)
```

Accelerate Java SE Libraries with CUDA

CUDA4J

CUDA C++ Programming Via Java APIs

Java 8

Accelerate Pure Java

IBM Developer Kits for Java: ibm.com/java/jdk

CUDA4J: GPU PROGRAMMING IN A JAVA API

Access CUDA Programming Model with Java Best Practices

```
void add(int[] a, int[] b, int[] c) throws CudaException, IOException {
  CudaDevice device = new CudaDevice(0);
  CudaModule module = new CudaModule(device,
                              getClass().getResourceAsStream("ArrayAdder"));
  CudaKernel kernel = new CudaKernel(module, "Cuda_cuda4j_samples_adder");

  CudaGrid grid = new CudaGrid(512, 512);

  try (CudaBuffer aBuffer = new CudaBuffer(device, a.length * 4);
       CudaBuffer bBuffer = new CudaBuffer(device, b.length * 4)) {
    aBuffer.copyFrom(a, 0, a.length);
    bBuffer.copyFrom(b, 0, b.length);

    kernel.launch(grid, new CudaKernel.Parameters(aBuffer, bBuffer, a.length));

    aBuffer.copyTo(c, 0, a.length);
  }
}
```

Manage CUDA devices and kernels

Easily transfer data between Java heap and CUDA device

Simple, flexible kernel launch

ACCELERATING PURE JAVA ON GPUS

Java 8 Streams and Lambda Expressions

- Express computation as aggregate parallel operations on data streams
- `IntStream.range(0, N).parallel().forEach(i -> c[i] = a[i] + b[i]);`
- Benefits
  - Standard Java idioms, so no code changes required
  - No knowledge of GPU programming model required
  - No low-level device manipulation – Java implementation has the controls
  - Future JIT smarts do not require application code changes

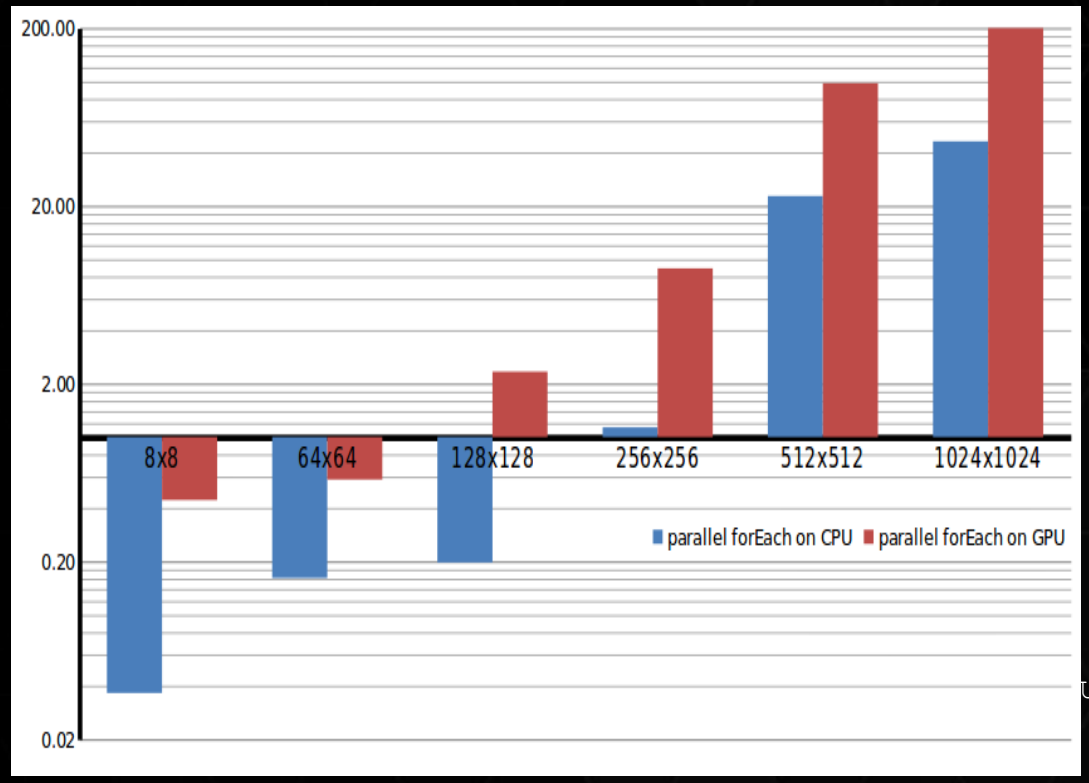
JIT / GPU OPTIMIZATION OF LAMBDA EXPRESSION

JIT-recognized Java matrix multiplication

```
Public void multiply() {
  IntStream.range(0, COLS*COLS).parallel().forEach(
    id -> {
      int i = id / COLS;
      int j = id % COLS;
      int sum = 0;

      for (int k = 0; k < COLS; k++) {
        sum += left[i*COLS + k] * right[k*COLS + j];
      }
      output[i*COLS + j] = sum;
  });
}
```

Speed-up factor when run on a GPU enabled host



COMMON PROGRAMMING APPROACHES

Across a Variety of Heterogeneous Systems

Libraries

AmgX, cuDNN, cuBLAS, OpenCV, Thrust

Compiler Directives

OpenACC

Programming Languages

C/C++, Fortran, python, Java